02 Amazon Fine Food Reviews Analysis_TSNE

April 12, 2019

1 Amazon Fine Food Reviews Analysis

Data Source: https://www.kaggle.com/snap/amazon-fine-food-reviews

EDA: https://nycdatascience.com/blog/student-works/amazon-fine-foods-visualization/

The Amazon Fine Food Reviews dataset consists of reviews of fine foods from Amazon.

Number of reviews: 568,454 Number of users: 256,059 Number of products: 74,258 Timespan: Oct 1999 - Oct 2012 Number of Attributes/Columns in data: 10

Attribute Information:

1. Id
2. ProductId - unique identifier for the product
3. UserId - unqiue identifier for the user
4. ProfileName
5. HelpfulnessNumerator - number of users who found the review helpful
6. HelpfulnessDenominator - number of users who indicated whether they found the review helpful or not
7. Score - rating between 1 and 5
8. Time - timestamp for the review
9. Summary - brief summary of the review
10. Text - text of the review

Objective: Given a review, determine whether the review is positive (Rating of 4 or 5) or negative (rating of 1 or 2).

[Q] How to determine if a review is positive or negative? [Ans] We could use the Score/Rating. A rating of 4 or 5 could be cosnidered a positive review. A review of 1 or 2 could be considered negative. A review of 3 is nuetral and ignored. This is an approximate and proxy way of determining the polarity (positivity/negativity) of a review.

1.1 Loading the data

The dataset is available in two forms 1. .csv file 2. SQLite Database

In order to load the data, We have used the SQLITE dataset as it easier to query the data and visualise the data efficiently.

Here as we only want to get the global sentiment of the recommendations (positive or negative), we will purposefully ignore all Scores equal to 3. If the score id above 3, then the recommendation wil be set to "positive". Otherwise, it will be set to "negative".

```
In [2]: %matplotlib inline
        import warnings
        warnings.filterwarnings("ignore")


        from gensim.models import Word2Vec
        from gensim.models import KeyedVectors

        import sqlite3
        import pandas as pd
        import numpy as np
        import nltk
        import string
        import matplotlib.pyplot as plt
        import seaborn as sns
        from sklearn.feature_extraction.text import TfidfTransformer
        from sklearn.feature_extraction.text import TfidfVectorizer

        from sklearn.feature_extraction.text import CountVectorizer
        from sklearn.metrics import confusion_matrix
        from sklearn import metrics
        from sklearn.metrics import roc_curve, auc
        from nltk.stem.porter import PorterStemmer

        import re
        # Tutorial about Python regular expressions: https://pymotw.com/2/re/
        import string
        from nltk.corpus import stopwords
        from nltk.stem import PorterStemmer
        from nltk.stem.wordnet import WordNetLemmatizer


        import pickle

        from tqdm import tqdm
        import os
```

2 [1]. Reading Data

```
In [3]: con = sqlite3.connect('database.sqlite')
        cursor = con.cursor()
        cursor.execute("SELECT name FROM sqlite_master WHERE type='table';")
        print(cursor.fetchall())
```

```
[('Reviews',)]
```

```
In [4]: # using the SQLite Table to read data.
```

```python
con = sqlite3.connect('database.sqlite')

#filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3
# SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000, will give top 500000 data point
# you can change the number to any other number based on your computing power

# filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 5
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 500

# Give reviews with Score>3 a positive rating, and reviews with a score<3 a negative r
def partition(x):
    if x < 3:
        return 0
    return 1

#changing reviews with score less than 3 to be positive and vice-versa
actualScore = filtered_data['Score']
positiveNegative = actualScore.map(partition)
filtered_data['Score'] = positiveNegative
print("Number of data points in our data", filtered_data.shape)
filtered_data.head(3)
```

```
Number of data points in our data (5000, 10)
```

Out[4]:

```
   Id   ProductId          UserId                      ProfileName  \
0   1  B001E4KFG0  A3SGXH7AUHU8GW                       delmartian
1   2  B00813GRG4  A1D87F6ZCVE5NK                           dll pa
2   3  B000LQOCH0   ABXLMWJIXXAIN  Natalia Corres "Natalia Corres"

   HelpfulnessNumerator  HelpfulnessDenominator  Score        Time  \
0                     1                       1      1  1303862400
1                     0                       0      0  1346976000
2                     1                       1      1  1219017600

                 Summary                                               Text
0  Good Quality Dog Food  I have bought several of the Vitality canned d...
1      Not as Advertised  Product arrived labeled as Jumbo Salted Peanut...
2  "Delight" says it all  This is a confection that has been around a fe...
```

In [5]:

```python
#this gives all the records where the exact same review by a user was found more than
display = pd.read_sql_query("""
SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
FROM Reviews
GROUP BY UserId
```

```
HAVING COUNT(*)>1
""", con)
```

In [6]:
```
print(display.shape)
display.head()
```

```
(80668, 7)
```

Out[6]:
```
                UserId   ProductId              ProfileName        Time  Score  \
0  #oc-R115TNMSPFT9I7  B007Y59HVM                  Breyton  1331510400      2
1  #oc-R11D9D7SHXIJB9  B005HG9ET0  Louis E. Emory "hoppy"  1342396800      5
2  #oc-R11DNU2NBKQ23Z  B007Y59HVM         Kim Cieszykowski  1348531200      1
3  #oc-R11O5J5ZVQE25C  B005HG9ET0            Penguin Chick  1346889600      5
4  #oc-R12KPBODL2B5ZD  B007OSBE1U    Christopher P. Presta  1348617600      1

                                                Text  COUNT(*)
0  Overall its just OK when considering the price...         2
1  My wife has recurring extreme muscle spasms, u...         3
2  This coffee is horrible and unfortunately not ...         2
3  This will be the bottle that you grab from the...         3
4  I didnt like this coffee. Instead of telling y...         2
```

In [7]:
```
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[7]:
```
              UserId   ProductId                       ProfileName        Time  \
80638  AZY10LLTJ71NX  B006P7E5ZI  undertheshrine "undertheshrine"  1334707200

       Score                                               Text  COUNT(*)
80638      5  I was recommended to try green tea extract to ...         5
```

In [8]:
```
display['COUNT(*)'].sum()
```

Out[8]:
```
393063
```

In [9]:
```
filtered_data.head(100)
filtered_data[filtered_data['UserId'] == 'A1D87F6ZCVE5NK']
```

Out[9]:
```
   Id   ProductId          UserId ProfileName  HelpfulnessNumerator  \
1   2  B00813GRG4  A1D87F6ZCVE5NK      dll pa                     0

   HelpfulnessDenominator  Score        Time             Summary  \
1                       0      0  1346976000  Not as Advertised

                                                Text
1  Product arrived labeled as Jumbo Salted Peanut...
```

3 Exploratory Data Analysis

3.1 [2] Data Cleaning: Deduplication

It is observed (as shown in the table below) that the reviews data had many duplicate entries. Hence it was necessary to remove duplicates in order to get unbiased results for the analysis of the data. Following is an example:

```
In [10]: display= pd.read_sql_query("""
         SELECT *
         FROM Reviews
         WHERE Score != 3 AND UserId="AR5J8UI46CURR"
         ORDER BY ProductID
         """, con)
         display.head()
```

```
Out[10]:        Id   ProductId         UserId      ProfileName  HelpfulnessNumerator  \
         0   78445  B000HDL1RQ  AR5J8UI46CURR  Geetha Krishnan                     2
         1  138317  B000HDOPYC  AR5J8UI46CURR  Geetha Krishnan                     2
         2  138277  B000HDOPYM  AR5J8UI46CURR  Geetha Krishnan                     2
         3   73791  B000HDOPZG  AR5J8UI46CURR  Geetha Krishnan                     2
         4  155049  B000PAQ75C  AR5J8UI46CURR  Geetha Krishnan                     2

            HelpfulnessDenominator  Score        Time  \
         0                       2      5  1199577600
         1                       2      5  1199577600
         2                       2      5  1199577600
         3                       2      5  1199577600
         4                       2      5  1199577600

                                      Summary  \
         0  LOACKER QUADRATINI VANILLA WAFERS
         1  LOACKER QUADRATINI VANILLA WAFERS
         2  LOACKER QUADRATINI VANILLA WAFERS
         3  LOACKER QUADRATINI VANILLA WAFERS
         4  LOACKER QUADRATINI VANILLA WAFERS

                                                       Text
         0  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
         1  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
         2  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
         3  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
         4  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
```

As can be seen above the same user has multiple reviews of the with the same values for HelpfulnessNumerator, HelpfulnessDenominator, Score, Time, Summary and Text and on doing analysis it was found that ProductId=B000HDOPZG was Loacker Quadratini Vanilla Wafer Cookies, 8.82-Ounce Packages (Pack of 8) ProductId=B000HDL1RQ was Loacker Quadratini Lemon Wafer Cookies, 8.82-Ounce Packages (Pack of 8) and so on

It was inferred after analysis that reviews with same parameters other than ProductId belonged to the same product just having different flavour or quantity. Hence in order to reduce redundancy it was decided to eliminate the rows having same parameters.

The method used for the same was that we first sort the data according to ProductId and then just keep the first similar product review and delelte the others. for eg. in the above just the review for ProductId=B000HDL1RQ remains. This method ensures that there is only one representative for each product and deduplication without sorting would lead to possibility of different representatives still existing for the same product.

In [11]:
```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=Fal
```

In [12]:
```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep=
final.shape
```

Out[12]:
```
(4986, 10)
```

In [13]:
```
#Checking to see how much % of data still remains
(final['Id'].size)/(filtered_data['Id'].size)*100
```

Out[13]:
```
99.72
```

Observation:- It was also seen that in two rows given below the value of HelpfulnessNumerator is greater than HelpfulnessDenominator which is not practically possible hence these two rows too are removed from calcualtions

In [14]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

Out[14]:
```
      Id   ProductId          UserId              ProfileName  \
0  64422  B000MIDROQ  A161DK06JJMCYF  J. E. Stephens "Jeanne"
1  44737  B001EQ55RW  A2V0I904FH7ABY                      Ram

   HelpfulnessNumerator  HelpfulnessDenominator  Score        Time  \
0                     3                       1      5  1224892800
1                     3                       2      4  1212883200

                                      Summary  \
0             Bought This for My Son at College
1  Pure cocoa taste with crunchy almonds inside

                                          Text
0  My son loves spaghetti so I didn't hesitate or...
1  It was almost a 'love at first bite' - the per...
```

In [15]:
```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [16]:
```
#Before starting the next phase of preprocessing lets see the number of entries left
print(final.shape)

#How many positive and negative reviews are present in our dataset?
final['Score'].value_counts()
```

```
(4986, 10)
```

Out[16]:
```
1    4178
0     808
Name: Score, dtype: int64
```

4 [3]. Text Preprocessing.

Now that we have finished deduplication our data requires some preprocessing before we go on further with analysis and making the prediction model.

Hence in the Preprocessing phase we do the following in the order below:-

1. Begin by removing the html tags
2. Remove any punctuations or limited set of special characters like , or . or # etc.
3. Check if the word is made up of english letters and is not alpha-numeric
4. Check to see if the length of the word is greater than 2 (as it was researched that there is no adjective in 2-letters)
5. Convert the word to lowercase
6. Remove Stopwords
7. Finally Snowball Stemming the word (it was obsereved to be better than Porter Stemming)

After which we collect the words used to describe positive and negative reviews

In [17]:
```
# printing some random reviews
sent_0 = final['Text'].values[0]
print(sent_0)
print("="*50)

sent_1000 = final['Text'].values[1000]
print(sent_1000)
print("="*50)

sent_1500 = final['Text'].values[1500]
print(sent_1500)
print("="*50)

sent_4900 = final['Text'].values[4900]
print(sent_4900)
print("="*50)
```

```
Why is this $[...] when the same product is available for $[...] here?<br />http://www.amazon.c
==================================================
I recently tried this flavor/brand and was surprised at how delicious these chips are.  The bes
==================================================
Wow.  So far, two two-star reviews.  One obviously had no idea what they were ordering; the otl
==================================================
love to order my coffee on amazon.  easy and shows up quickly.<br />This k cup is great coffee
==================================================
```

```
In [18]: # remove urls from text python: https://stackoverflow.com/a/40823105/4084039
         sent_0 = re.sub(r"http\S+", "", sent_0)
         sent_1000 = re.sub(r"http\S+", "", sent_1000)
         sent_150 = re.sub(r"http\S+", "", sent_1500)
         sent_4900 = re.sub(r"http\S+", "", sent_4900)

         print(sent_0)
```

```
Why is this $[...] when the same product is available for $[...] here?<br /> /><br />The Victor
```

```
In [19]: # https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all-
         from bs4 import BeautifulSoup

         soup = BeautifulSoup(sent_0, 'lxml')
         text = soup.get_text()
         print(text)
         print("="*50)

         soup = BeautifulSoup(sent_1000, 'lxml')
         text = soup.get_text()
         print(text)
         print("="*50)

         soup = BeautifulSoup(sent_1500, 'lxml')
         text = soup.get_text()
         print(text)
         print("="*50)

         soup = BeautifulSoup(sent_4900, 'lxml')
         text = soup.get_text()
         print(text)
```

```
Why is this $[...] when the same product is available for $[...] here? />The Victor M380 and M5
==================================================
I recently tried this flavor/brand and was surprised at how delicious these chips are.  The bes
==================================================
Wow.  So far, two two-star reviews.  One obviously had no idea what they were ordering; the otl
==================================================
```

```
love to order my coffee on amazon.  easy and shows up quickly.This k cup is great coffee.  dca
```

```
In [20]: # https://stackoverflow.com/a/47091490/4084039
         import re

         def decontracted(phrase):
             # specific
             phrase = re.sub(r"won't", "will not", phrase)
             phrase = re.sub(r"can\'t", "can not", phrase)

             # general
             phrase = re.sub(r"n\'t", " not", phrase)
             phrase = re.sub(r"\'re", " are", phrase)
             phrase = re.sub(r"\'s", " is", phrase)
             phrase = re.sub(r"\'d", " would", phrase)
             phrase = re.sub(r"\'ll", " will", phrase)
             phrase = re.sub(r"\'t", " not", phrase)
             phrase = re.sub(r"\'ve", " have", phrase)
             phrase = re.sub(r"\'m", " am", phrase)
             return phrase
```

```
In [21]: sent_1500 = decontracted(sent_1500)
         print(sent_1500)
         print("="*50)
```

```
Wow.  So far, two two-star reviews.  One obviously had no idea what they were ordering; the ot
==================================================
```

```
In [22]: #remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
         sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
         print(sent_0)
```

```
Why is this $[...] when the same product is available for $[...] here?<br /> /><br />The Victo
```

```
In [23]: #remove spacial character: https://stackoverflow.com/a/5843547/4084039
         sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
         print(sent_1500)
```

```
Wow So far two two star reviews One obviously had no idea what they were ordering the other wa
```

```
In [24]: # https://gist.github.com/sebleier/554280
         # we are removing the words from the stop words list: 'no', 'nor', 'not'
         # <br /><br /> ==> after the above steps, we are getting "br br"
         # we are including them into stop words list
         # instead of <br /> if we have <br/> these tags would have revmoved in the 1st step
```

```
stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselve
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him'
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself',
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'h
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'o
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'an
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'n
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't"
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mig
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't",
            'won', "won't", 'wouldn', "wouldn't"])
```

In [25]:
```
# Combining all the above stundents
#tqdm is the progress bar
from tqdm import tqdm
sno = nltk.stem.SnowballStemmer('english')

preprocessed_reviews = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Text'].values):
    #remove URLs
    sentance = re.sub(r"http\S+", "", sentance)
    #remove hml tags
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    # decontract : won't -> will not
    sentance = decontracted(sentance)
    # remove words with numbers : eg abc123  or just 1234 are both filtered out
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    # remove special characters
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    # https://gist.github.com/sebleier/554280
    # also performing stemming here using Snowball stemmer.
    #if we stem then  pre-trained Google W2V may fail to find a vector for tasti (the
    #sentance = ' '.join(sno.stem(e.lower()) for e in sentance.split() if e.lower() no
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in stopwo
    preprocessed_reviews.append(sentance.strip())
```

```
100%|| 4986/4986 [00:01<00:00, 2621.27it/s]
```

In [26]:
```
preprocessed_reviews[1500]
len(preprocessed_reviews[1500].split())
```

Out[26]: 129

[3.2] Preprocess Summary

```
In [27]: ## Similarly you can do preprocessing for review summary also.
         #print some random summary values

         summary_0 = final['Summary'].values[0]
         print(summary_0)
         print("="*50)

         summary_1000 = final['Summary'].values[1000]
         print(summary_1000)
         print("="*50)

         summary_4000 = final['Summary'].values[4000]
         print(summary_4000)
         print("="*50)
```

```
thirty bucks?
==================================================
Best sour cream & onion chip I've had
==================================================
Pleasantly surprised
==================================================
```

```
In [28]: #preprocessed summary

         preprocessed_summaries = []
         # tqdm is for printing the status bar
         for sentence in tqdm(final['Summary'].values):
             #remove URLs
             sentence = re.sub(r"http\S+", "", sentence)
             #remove hml tags
             sentence = BeautifulSoup(sentence, 'lxml').get_text()
             # decontract : won't -> will not
             sentence = decontracted(sentence)
             # remove words with numbers : eg abc123  or just 1234 are both filtered out
             sentence = re.sub("\S*\d\S*", "", sentence).strip()
             # remove special characters
             # if we do not do the step above then this one will convert an 'abc123' to an 'ab
             # the above step will ensure that abc123 is completely removed from our result se
             sentence = re.sub('[^A-Za-z]+', ' ', sentence)
             # https://gist.github.com/sebleier/554280
             # also performing stemming here using Snowball stemmer.
             #if we stem then  pre-trained Google W2V may fail to find a vector for tasti (the
             #sentence = ' '.join(sno.stem(e.lower()) for e in sentence.split() if e.lower() no
             sentence = ' '.join(e.lower() for e in sentence.split() if e.lower() not in stopwo
             preprocessed_summaries.append(sentence.strip())
```

```
100%|| 4986/4986 [00:01<00:00, 3934.08it/s]
```

```
In [29]: #print the same pre_processed summary values

         summary_0 = preprocessed_summaries[0]
         print(summary_0)
         print("="*50)

         summary_1000 = preprocessed_summaries[1000]
         print(summary_1000)
         print("="*50)

         summary_4000 = preprocessed_summaries[4000]
         print(summary_4000)
         print("="*50)
```

```
thirty bucks
==================================================
best sour cream onion chip
==================================================
pleasantly surprised
==================================================
```

5 [4] Featurization

5.1 [4.1] BAG OF WORDS

```
In [30]: #BoW
         count_vect = CountVectorizer() #in scikit-learn
         count_vect.fit(preprocessed_reviews)
         print("some feature names ", count_vect.get_feature_names()[:10])
         print('='*50)

         final_counts = count_vect.transform(preprocessed_reviews)
         print("the type of count vectorizer ",type(final_counts))
         print("the shape of out text BOW vectorizer ",final_counts.get_shape())
         print("the number of unique words ", final_counts.get_shape()[1])
```

```
some feature names  ['aa', 'aahhhs', 'aback', 'abandon', 'abates', 'abbott', 'abby', 'abdomina
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (4986, 12997)
the number of unique words  12997
```

5.2 [4.2] Bi-Grams and n-Grams.

In [31]:
```
#bi-gram, tri-gram and n-gram

#removing stop words like "not" should be avoided before building n-grams
# count_vect = CountVectorizer(ngram_range=(1,2))
# please do read the CountVectorizer documentation http://scikit-learn.org/stable/mod
# you can choose these numebrs min_df=10, max_features=5000, of your choice
count_vect = CountVectorizer(ngram_range=(1,2), min_df=10, max_features=5000)
final_bigram_counts = count_vect.fit_transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_bigram_counts))
print("the shape of out text BOW vectorizer ",final_bigram_counts.get_shape())
print("the number of unique words including both unigrams and bigrams ", final_bigram
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (4986, 3144)
the number of unique words including both unigrams and bigrams  3144
```

5.3 [4.3] TF-IDF

In [81]:
```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(preprocessed_reviews)
print("some sample features(unique words in the corpus)",tf_idf_vect.get_feature_name

print('='*50)

final_tf_idf = tf_idf_vect.transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_tf_idf))
print("the shape of out text TFIDF vectorizer ",final_tf_idf.get_shape())
print("the number of unique words including both unigrams and bigrams ", final_tf_idf
final_tf_idf_dense = final_tf_idf.todense()
```

```
some sample features(unique words in the corpus) ['ability', 'able', 'able find', 'able get',
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (4986, 3144)
the number of unique words including both unigrams and bigrams  3144
```

5.4 [4.4] Word2Vec

In [33]:
```
# Train your own Word2Vec model using your own text corpus
i=0
list_of_sentance=[]
for sentance in preprocessed_reviews:
    list_of_sentance.append(sentance.split())
```

In [34]:
```
# Using Google News Word2Vectors
```

```
# in this project we are using a pretrained model by google
# its 3.3G file, once you load this into your memory
# it occupies ~9Gb, so please do this step only if you have >12G of ram
# we will provide a pickle file wich contains a dict ,
# and it contains all our courpus words as keys and  model[word] as values
# To use this code-snippet, download "GoogleNews-vectors-negative300.bin"
# from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit
# it's 1.9GB in size.


# http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17SRFAzZPY
# you can comment this whole cell
# or change these varible according to your need

is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True

if want_to_train_w2v:
    # min_count = 5 considers only words that occured atleast 5 times
    w2v_model=Word2Vec(list_of_sentance,min_count=5,size=50, workers=4)
    print(w2v_model.wv.most_similar('great'))
    print('='*50)
    print(w2v_model.wv.most_similar('worst'))

elif want_to_use_google_w2v and is_your_ram_gt_16g:
    if os.path.isfile('GoogleNews-vectors-negative300.bin'):
        w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bi
        print(w2v_model.wv.most_similar('great'))
        print(w2v_model.wv.most_similar('worst'))
    else:
        print("you don't have gogole's word2vec file, keep want_to_train_w2v = True, t
```

```
[('excellent', 0.9966654777526855), ('wonderful', 0.9961280226707458), ('think', 0.996052563190
==================================================
[('part', 0.999345600605011), ('kitchen', 0.9993090629577637), ('grown', 0.9992631673812866),
```

In [35]:
```
w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words))
print("sample words ", w2v_words[0:50])
```

```
number of words that occured minimum 5 times  3817
sample words  ['product', 'available', 'course', 'total', 'pretty', 'stinky', 'right', 'nearby
```

5.5 [4.4.1] Converting text into vectors using wAvg W2V, TFIDF-W2V

[4.4.1.1] Avg W2v

In [36]:
```
# average Word2Vec
# compute average word2vec for each review.
sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need t
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors.append(sent_vec)
print(len(sent_vectors))
print(len(sent_vectors[0]))
```

```
100%|| 4986/4986 [00:04<00:00, 1146.71it/s]

4986
50
```

[4.4.1.2] TFIDF weighted W2v

In [37]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
model.fit(preprocessed_reviews)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

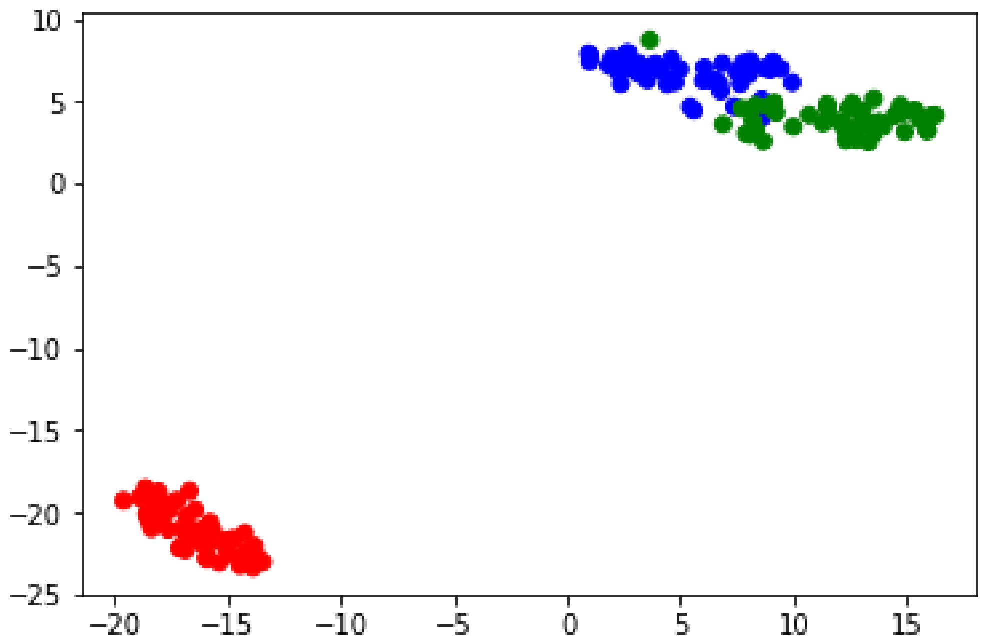
In [38]:
```
# TF-IDF weighted Word2Vec
tfidf_feat = model.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this l
row=0;
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
#             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
```

```
                # sent.count(word) = tf valeus of word in this review
                tf_idf = dictionary[word]*(sent.count(word)/len(sent))
                sent_vec += (vec * tf_idf)
                weight_sum += tf_idf
        if weight_sum != 0:
            sent_vec /= weight_sum
        tfidf_sent_vectors.append(sent_vec)
        row += 1
```

```
100%|| 4986/4986 [00:24<00:00, 204.13it/s]
```

6 [5] Applying TSNE

In [39]:
```
# https://github.com/pavlin-policar/fastTSNE you can try this also, this version is l
import numpy as np
from sklearn.manifold import TSNE
from sklearn import datasets
import pandas as pd
import matplotlib.pyplot as plt

iris = datasets.load_iris()
x = iris['data']
y = iris['target']

tsne = TSNE(n_components=2, perplexity=30, learning_rate=200)

X_embedding = tsne.fit_transform(x)
# if x is a sparse matrix you need to pass it as X_embedding = tsne.fit_transform(x.t
for_tsne = np.hstack((X_embedding, y.reshape(-1,1)))
for_tsne_df = pd.DataFrame(data=for_tsne, columns=['Dimension_x','Dimension_y','Score
colors = {0:'red', 1:'blue', 2:'green'}
plt.scatter(for_tsne_df['Dimension_x'], for_tsne_df['Dimension_y'], c=for_tsne_df['Sc
plt.show()
```



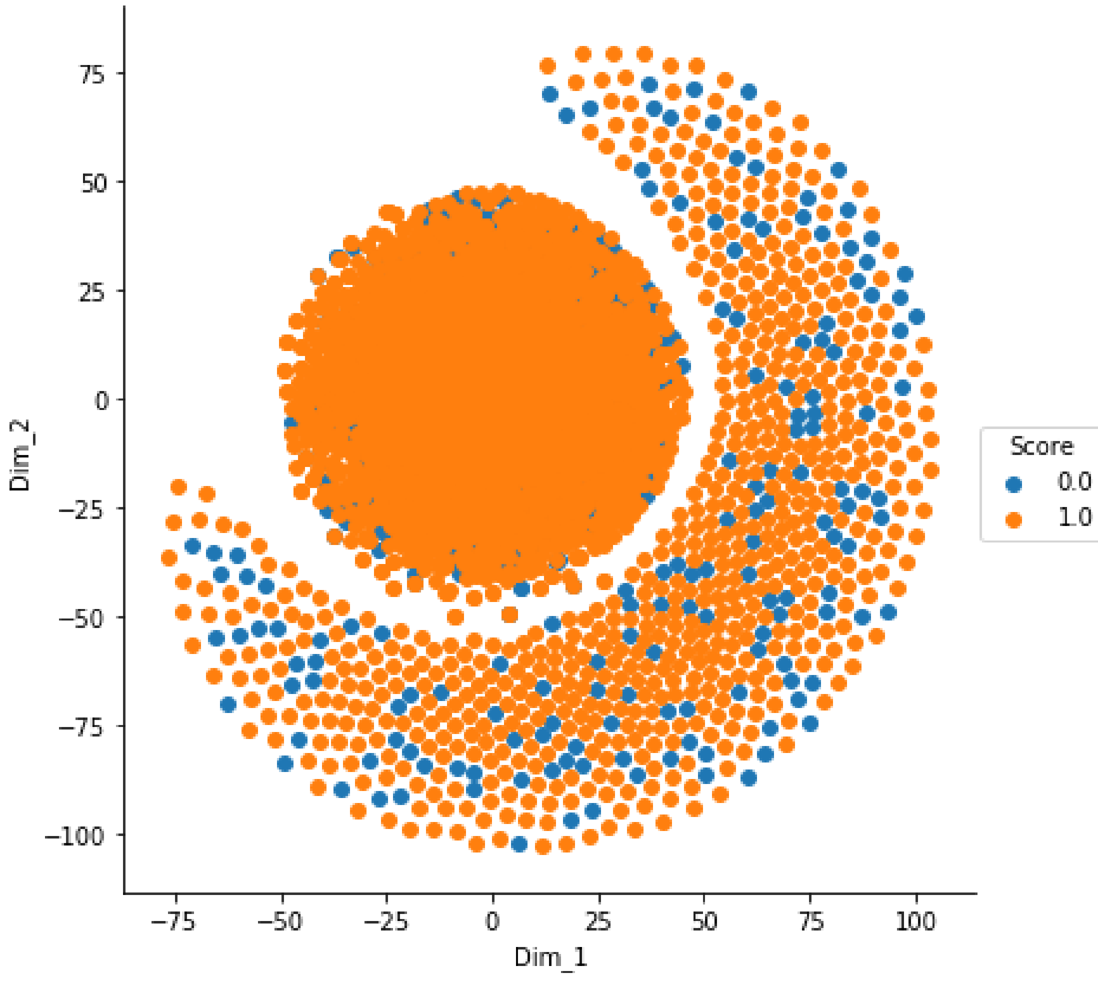

```
<li> you need to plot 4 tsne plots with each of these feature set
    <ol>
        <li>Review text, preprocessed one converted into vectors using (BOW)</li>
        <li>Review text, preprocessed one converted into vectors using (TFIDF)</li>
        <li>Review text, preprocessed one converted into vectors using (AVG W2v)</li>
        <li>Review text, preprocessed one converted into vectors using (TFIDF W2v)</li>
    </ol>
</li>
<li> <font color='blue'>Note 1: The TSNE accepts only dense matrices</font></li>
<li> <font color='blue'>Note 2: Consider only 5k to 6k data points </font></li>
```

6.1 [5.1] Applying TNSE on Text BOW vectors

In [50]:
```python
import time
from sklearn.manifold import TSNE
from sklearn.preprocessing import StandardScaler
import seaborn as sn

start = time.time()
# final_counts has the BOW vectors but is a sparse matrix
bow_vectors_dense = final_counts.todense()
standardized_data = StandardScaler().fit_transform(bow_vectors_dense)
print(standardized_data.shape)

tsne = TSNE(n_components=2, perplexity=50, learning_rate=200)
X_embedding = tsne.fit_transform(standardized_data)
```

```
end = time.time()
print('Time Taken:' , end - start)
print(X_embedding.shape)
Y = final['Score']
tsne_data = np.vstack((X_embedding.T, Y)).T
tsne_df = pd.DataFrame(data=tsne_data, columns=("Dim_1", "Dim_2", "Score"))
# Ploting the result of tsne
sn.FacetGrid(tsne_df, hue="Score", size=6).map(plt.scatter, 'Dim_1', 'Dim_2').add_lege
plt.show()
```

```
(4986, 12997)
570.3803977966309
(4986, 2)
```

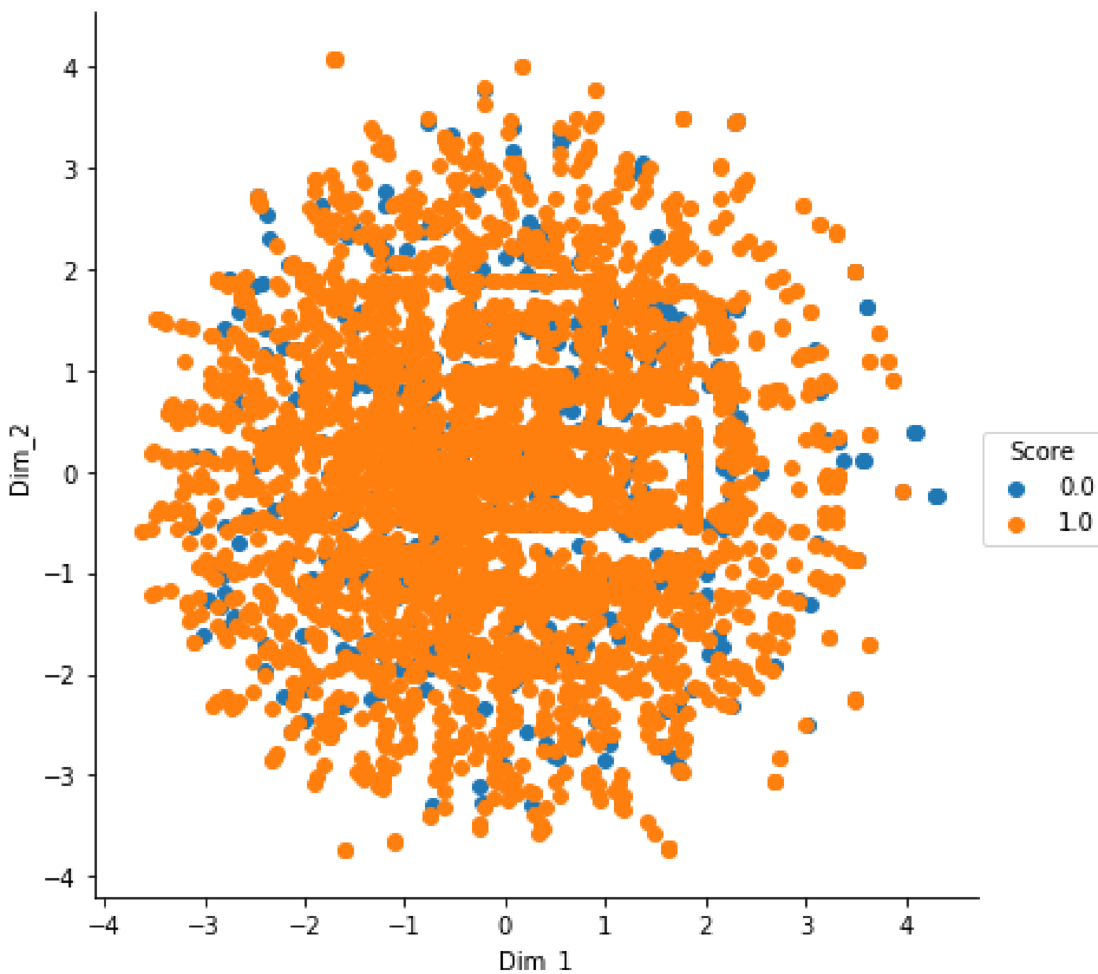



```
In [54]: # Code from : https://github.com/pavlin-policar/fastTSNE
```

```
import time
from openTSNE import TSNE
from openTSNE.callbacks import ErrorLogger
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
import seaborn as sn

# final_counts has the BOW vectors but is a sparse matrix
bow_vectors_dense = final_counts.todense()
standardized_data = StandardScaler().fit_transform(bow_vectors_dense)
print(standardized_data.shape)
start = time.time()

tsne = TSNE(
    perplexity=30,
    metric="euclidean",
    callbacks=ErrorLogger(),
    n_jobs=8,
    random_state=42,
)

%time X_embedding = tsne.fit(standardized_data)
end = time.time()
print('Time Taken:' , end - start)

Y = final['Score']
tsne_data = np.vstack((X_embedding.T, Y)).T
tsne_df = pd.DataFrame(data=tsne_data, columns=("Dim_1", "Dim_2", "Score"))
# Ploting the result of tsne
sn.FacetGrid(tsne_df, hue="Score", size=6).map(plt.scatter, 'Dim_1', 'Dim_2').add_lege
plt.show()
```

```
(4986, 12997)
Iteration   50, KL divergence  9.4085, 50 iterations in 78.3106 sec
Iteration  100, KL divergence  9.7325, 50 iterations in 55.1784 sec
Iteration  150, KL divergence  10.2235, 50 iterations in 286.9073 sec
Iteration  200, KL divergence  8.4935, 50 iterations in 285.8731 sec
Iteration  250, KL divergence  9.6837, 50 iterations in 257.1334 sec
Iteration   50, KL divergence  4.0338, 50 iterations in 246.5137 sec
Iteration  100, KL divergence  3.7375, 50 iterations in 70.1215 sec
Iteration  150, KL divergence  3.4758, 50 iterations in 6.4602 sec
Iteration  200, KL divergence  3.4308, 50 iterations in 0.5254 sec
Iteration  250, KL divergence  3.4283, 50 iterations in 0.5379 sec
Iteration  300, KL divergence  3.4269, 50 iterations in 0.5272 sec
Iteration  350, KL divergence  3.4259, 50 iterations in 0.5126 sec
Iteration  400, KL divergence  3.4250, 50 iterations in 0.5177 sec
Iteration  450, KL divergence  3.4242, 50 iterations in 0.5064 sec
```

```
Iteration  500, KL divergence  3.4242, 50 iterations in 0.5454 sec
Iteration  550, KL divergence  3.4245, 50 iterations in 0.5403 sec
Iteration  600, KL divergence  3.4243, 50 iterations in 0.5133 sec
Iteration  650, KL divergence  3.4240, 50 iterations in 0.5192 sec
Iteration  700, KL divergence  3.4238, 50 iterations in 0.5245 sec
Iteration  750, KL divergence  3.4238, 50 iterations in 0.5260 sec
CPU times: user 29min 25s, sys: 1min 43s, total: 31min 9s
Wall time: 26min 54s
```

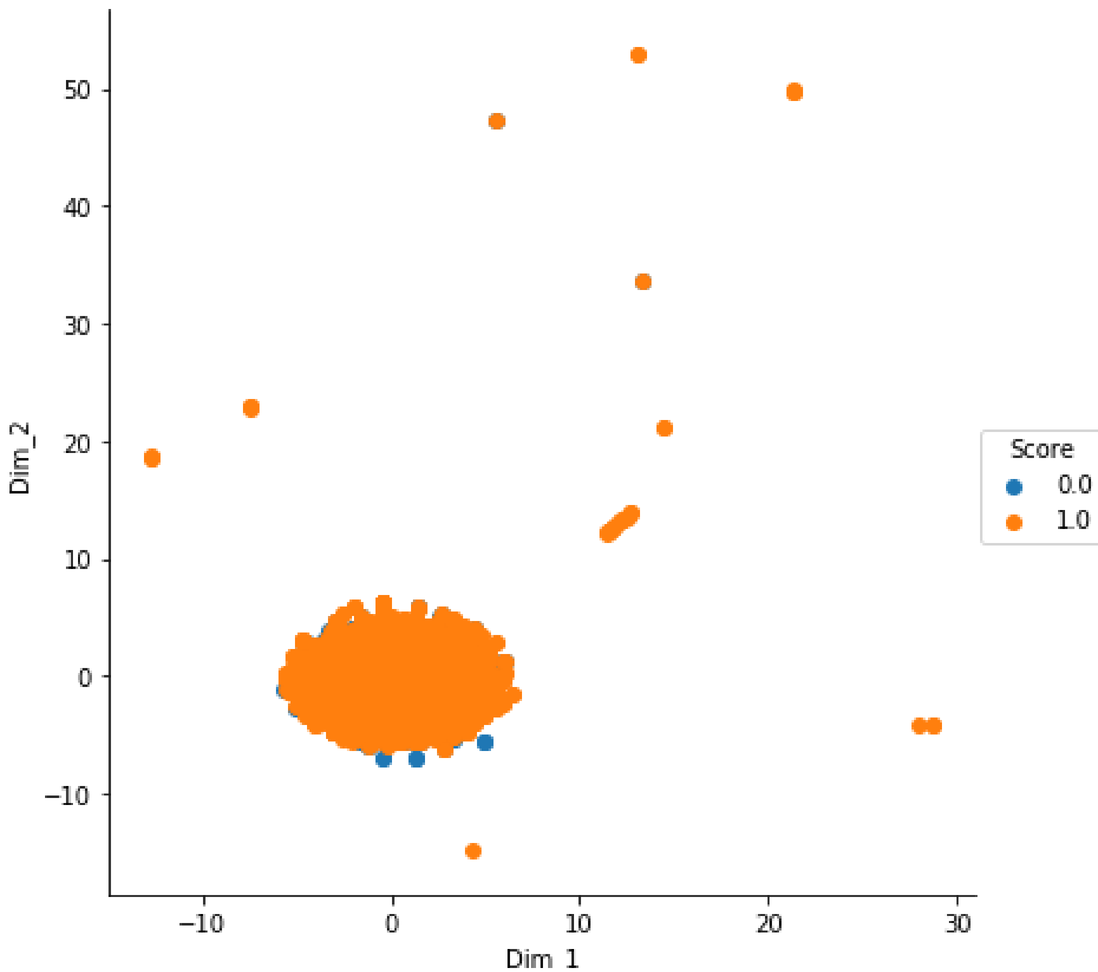


6.2 [5.1] Applying TNSE on Text TFIDF vectors

```
In [60]: import time
         from sklearn.manifold import TSNE
         from sklearn.preprocessing import StandardScaler
         import seaborn as sn
```

```python
# final_tf_idf has the TFIDF vectors but is a sparse matrix
final_tf_idf_dense = final_tf_idf.todense()
standardized_data = StandardScaler().fit_transform(final_tf_idf_dense)
print(standardized_data.shape)

start = time.time()

# If the learning rate is too low, most points may look compressed in a dense cloud w
# it happens here with learning_rate set to 500
tsne = TSNE(n_components=2, perplexity=30, learning_rate=500)
X_embedding = tsne.fit_transform(standardized_data)
end = time.time()
print('Time Taken:' , end - start)
print(X_embedding.shape)
Y = final['Score']
tsne_data = np.vstack((X_embedding.T, Y)).T
tsne_df = pd.DataFrame(data=tsne_data, columns=("Dim_1", "Dim_2", "Score"))
# Ploting the result of tsne
sn.FacetGrid(tsne_df, hue="Score", size=6).map(plt.scatter, 'Dim_1', 'Dim_2').add_lege
plt.show()
```

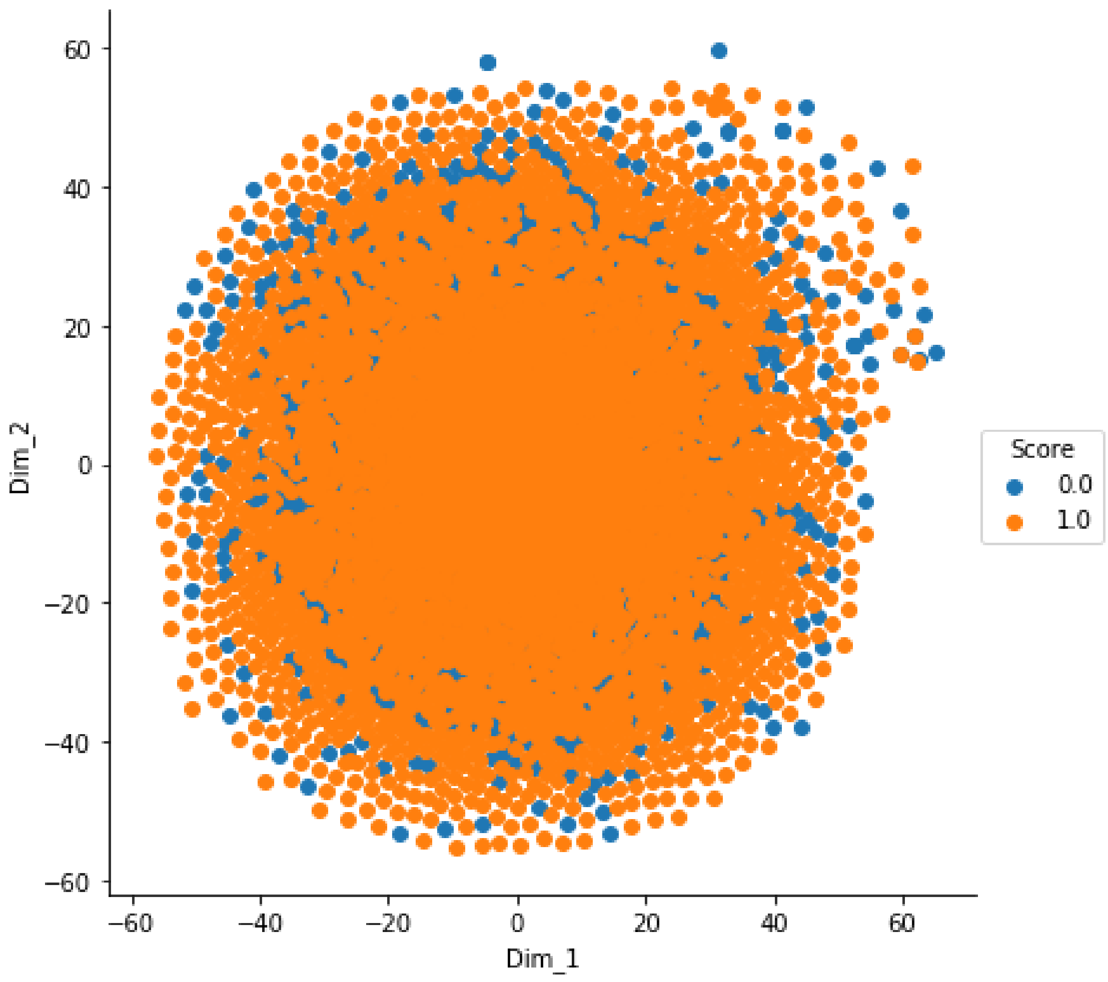
```
(4986, 3144)
Time Taken: 169.43431091308594
(4986, 2)
```




```
In [73]: import time
         from sklearn.manifold import TSNE
         from sklearn.preprocessing import StandardScaler
         import seaborn as sn



         # final_tf_idf has the TFIDF vectors but is a sparse matrix
         final_tf_idf_dense = final_tf_idf.todense()
         standardized_data = StandardScaler().fit_transform(final_tf_idf_dense)
         print(standardized_data.shape)

         start = time.time()

         # If the learning rate is too low, most points may look compressed in a dense cloud w
         # it happens here with learning_rate set to 200
```

```
#tsne = TSNE(n_components=2, perplexity=30, learning_rate=200)
tsne = TSNE(n_components=2, perplexity=500, learning_rate=2000)
X_embedding = tsne.fit_transform(standardized_data)
end = time.time()
print('Time Taken:' , end - start)
print(X_embedding.shape)
Y = final['Score']
tsne_data = np.vstack((X_embedding.T, Y)).T
tsne_df = pd.DataFrame(data=tsne_data, columns=("Dim_1", "Dim_2", "Score"))
# Ploting the result of tsne
sn.FacetGrid(tsne_df, hue="Score", size=6).map(plt.scatter, 'Dim_1', 'Dim_2').add_lege
plt.show()
```

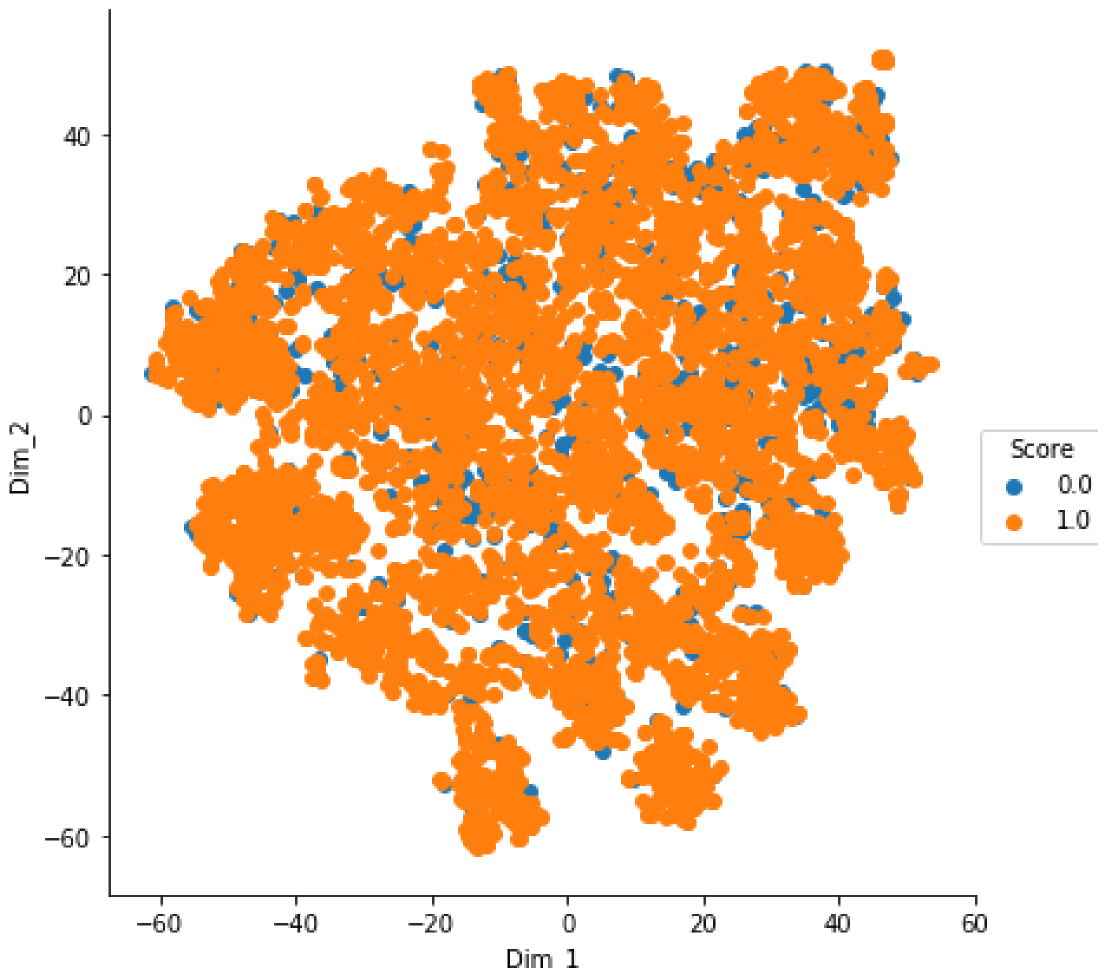
```
(4986, 3144)
Time Taken: 364.97389698028564
(4986, 2)
```



6.3 [5.3] Applying TNSE on Text Avg W2V vectors

In [59]:
```
import time
from sklearn.manifold import TSNE



standardized_avg_w2v_data = StandardScaler().fit_transform(sent_vectors)
print(standardized_avg_w2v_data.shape)

start = time.time()
tsne = TSNE(n_components=2, perplexity=50, learning_rate=500)
X_embedding = tsne.fit_transform(standardized_avg_w2v_data)
end = time.time()
print('Time Taken:' , end - start)
print(X_embedding.shape)
Y = final['Score']
tsne_data = np.vstack((X_embedding.T, Y)).T
tsne_df = pd.DataFrame(data=tsne_data, columns=("Dim_1", "Dim_2", "Score"))
# Ploting the result of tsne
sn.FacetGrid(tsne_df, hue="Score", size=6).map(plt.scatter, 'Dim_1', 'Dim_2').add_lege
plt.show()
```

```
(4986, 50)
Time Taken: 48.124521255493164
(4986, 2)
```

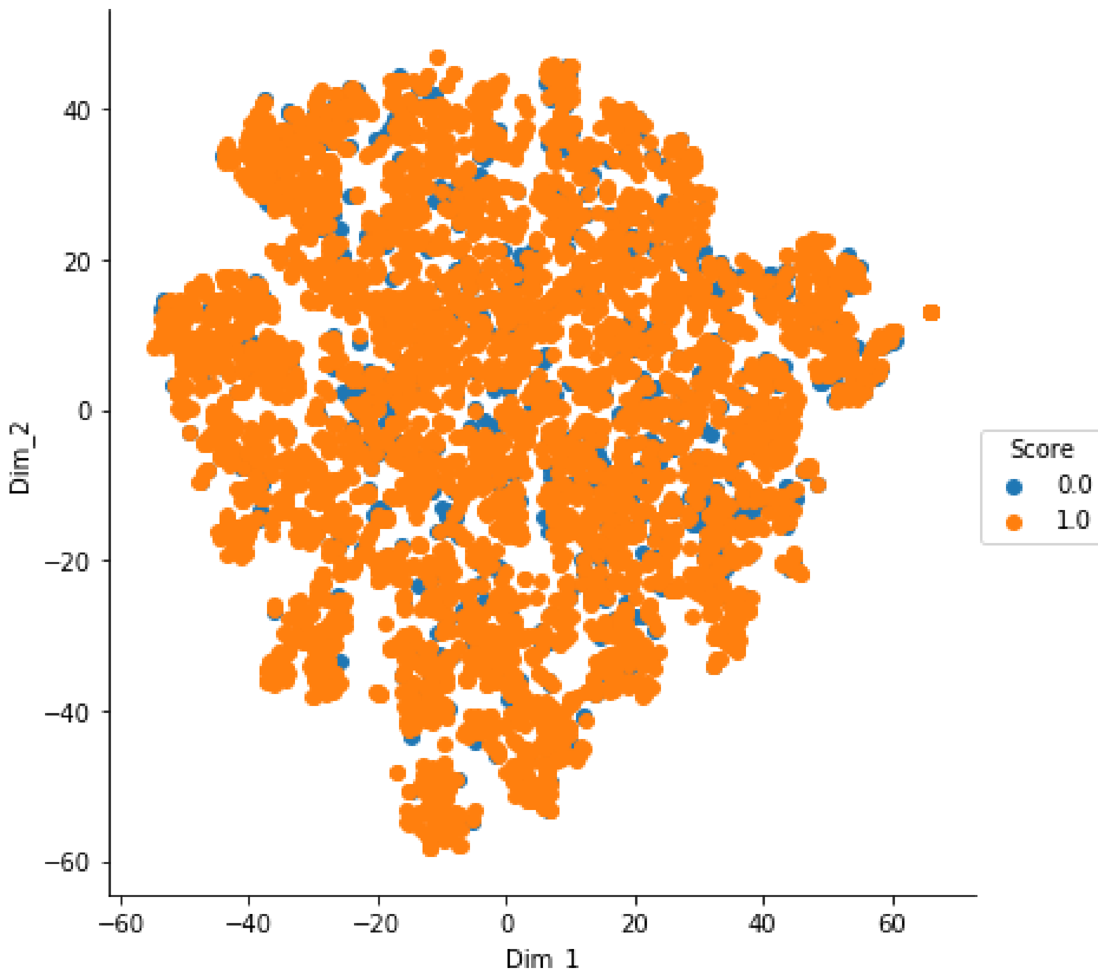


7 [5.3.1] Using OpenTSNE

```
In [58]: # code from : https://github.com/pavlin-policar/fastTSNE
         from openTSNE import TSNE
         from openTSNE.callbacks import ErrorLogger
         import time

         standardized_avg_w2v_data = StandardScaler().fit_transform(sent_vectors)
         print(standardized_avg_w2v_data.shape)

         start = time.time()
         tsne = TSNE(
             perplexity=30,
             metric="euclidean",
             callbacks=ErrorLogger(),
             n_jobs=8,
```

```
        random_state=42)

X_embedding = tsne.fit(standardized_avg_w2v_data)
end = time.time()
print('Time Taken:' , end - start)

print(X_embedding.shape)
Y = final['Score']
tsne_data = np.vstack((X_embedding.T, Y)).T
tsne_df = pd.DataFrame(data=tsne_data, columns=("Dim_1", "Dim_2", "Score"))
# Ploting the result of tsne
sn.FacetGrid(tsne_df, hue="Score", size=6).map(plt.scatter, 'Dim_1', 'Dim_2').add_lege
plt.show()
```

```
(4986, 50)
Iteration   50, KL divergence  4.1090, 50 iterations in 0.4368 sec
Iteration  100, KL divergence  4.0985, 50 iterations in 0.4397 sec
Iteration  150, KL divergence  4.0984, 50 iterations in 0.4420 sec
Iteration  200, KL divergence  4.0984, 50 iterations in 0.4350 sec
Iteration   50, KL divergence  2.1748, 50 iterations in 0.6092 sec
Iteration  100, KL divergence  1.8739, 50 iterations in 1.4441 sec
Iteration  150, KL divergence  1.7565, 50 iterations in 3.2114 sec
Iteration  200, KL divergence  1.7043, 50 iterations in 5.2291 sec
Iteration  250, KL divergence  1.6774, 50 iterations in 7.8610 sec
Iteration  300, KL divergence  1.6610, 50 iterations in 9.1923 sec
Iteration  350, KL divergence  1.6486, 50 iterations in 10.6554 sec
Iteration  400, KL divergence  1.6392, 50 iterations in 12.9989 sec
Iteration  450, KL divergence  1.6316, 50 iterations in 18.7020 sec
Iteration  500, KL divergence  1.6248, 50 iterations in 10.9947 sec
Iteration  550, KL divergence  1.6188, 50 iterations in 19.1845 sec
Iteration  600, KL divergence  1.6146, 50 iterations in 12.8447 sec
Iteration  650, KL divergence  1.6120, 50 iterations in 15.3767 sec
Iteration  700, KL divergence  1.6070, 50 iterations in 13.7797 sec
Iteration  750, KL divergence  1.6039, 50 iterations in 18.9356 sec
Time Taken: 167.64498209953308
(4986, 2)
```

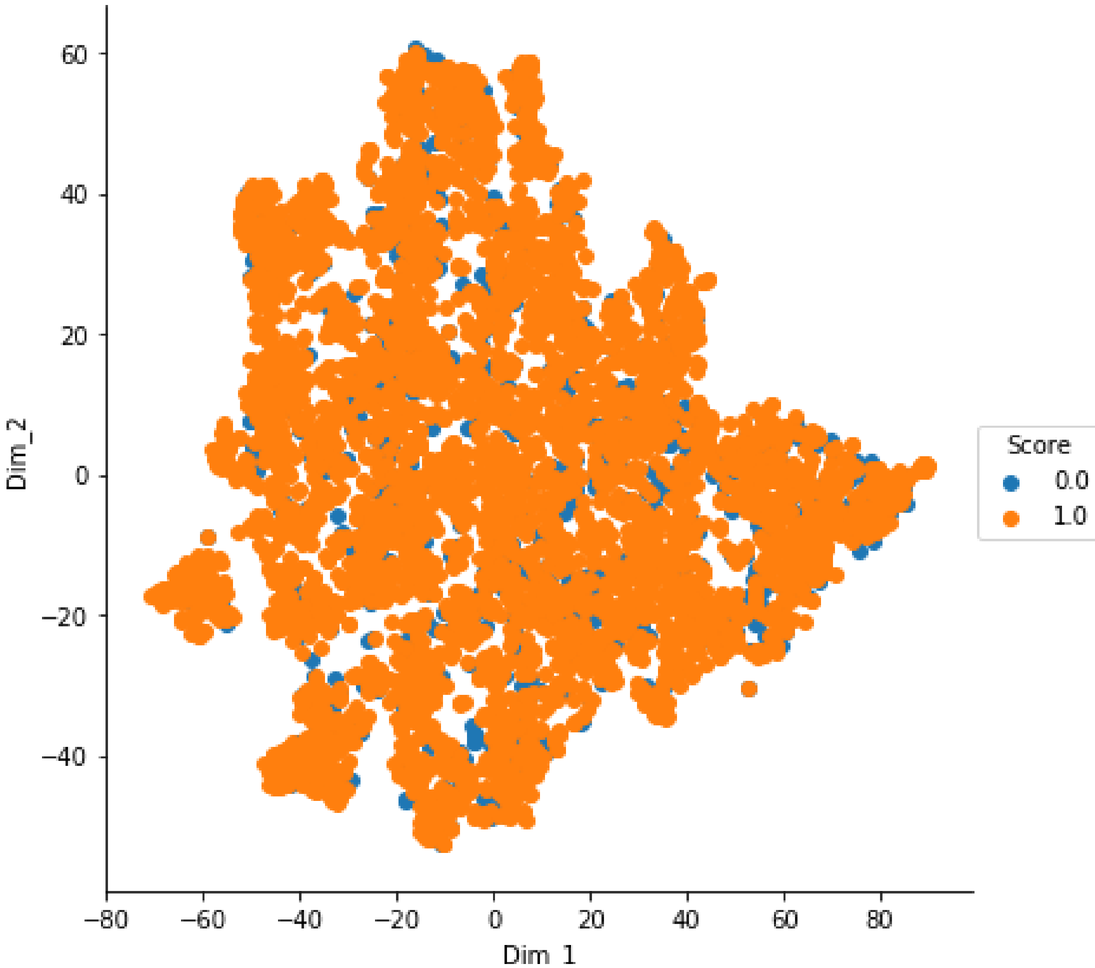


8 [5.4] Applying TNSE on Text TFIDF weighted W2V vectors

In [57]:
```
import time
from sklearn.manifold import TSNE

standardized_tfidf_w2v_data = StandardScaler().fit_transform(tfidf_sent_vectors)
print(standardized_tfidf_w2v_data.shape)

start = time.time()
tsne = TSNE(n_components=2, perplexity=50, learning_rate=500)
X_embedding = tsne.fit_transform(standardized_tfidf_w2v_data)
end = time.time()
print('Time Taken:' , end - start)
print(X_embedding.shape)
Y = final['Score']
tsne_data = np.vstack((X_embedding.T, Y)).T
```

```
tsne_df = pd.DataFrame(data=tsne_data, columns=("Dim_1", "Dim_2", "Score"))
# Ploting the result of tsne
sn.FacetGrid(tsne_df, hue="Score", size=6).map(plt.scatter, 'Dim_1', 'Dim_2').add_lege
plt.show()
```

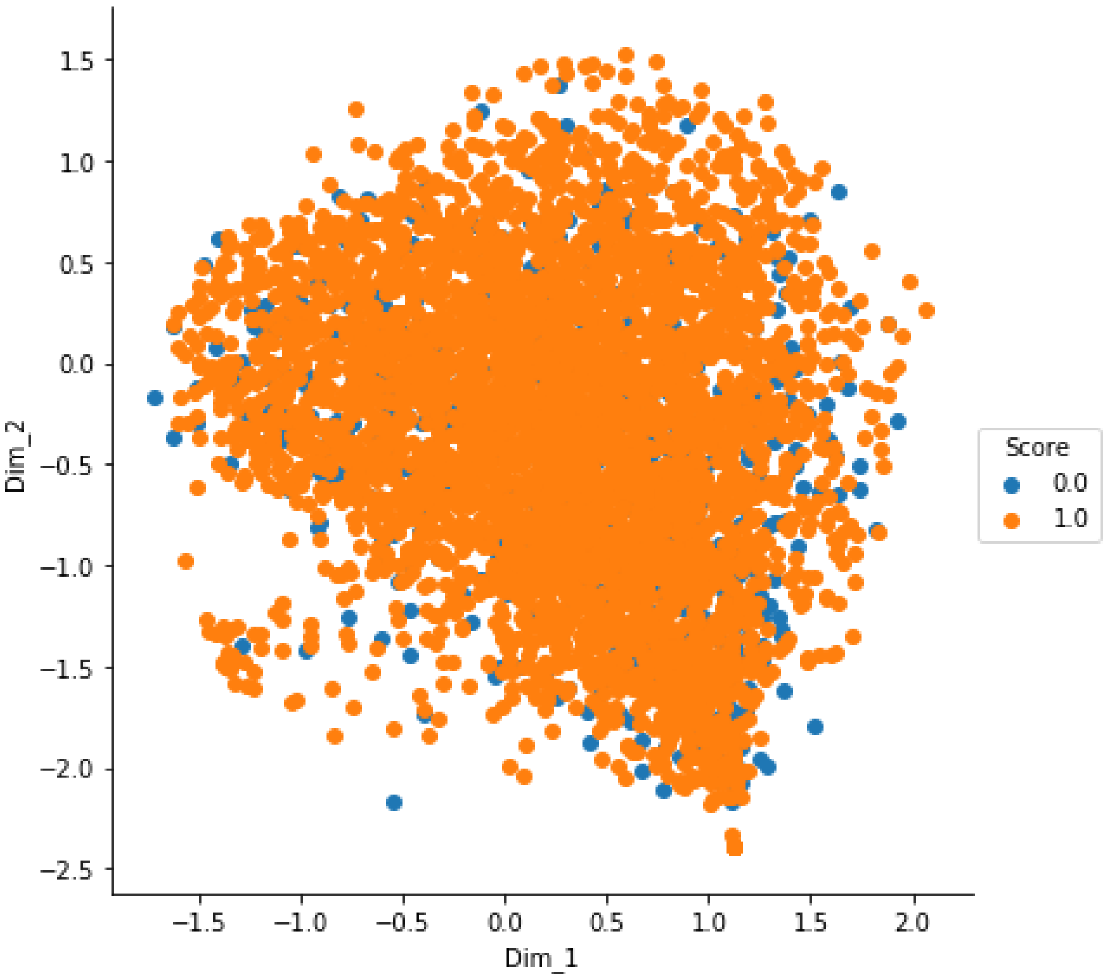
```
(4986, 50)
Time Taken: 52.18776488304138
(4986, 2)
```




```
In [79]: import time
         from sklearn.manifold import TSNE

         standardized_tfidf_w2v_data = StandardScaler().fit_transform(tfidf_sent_vectors)
         print(standardized_tfidf_w2v_data.shape)

         start = time.time()
```

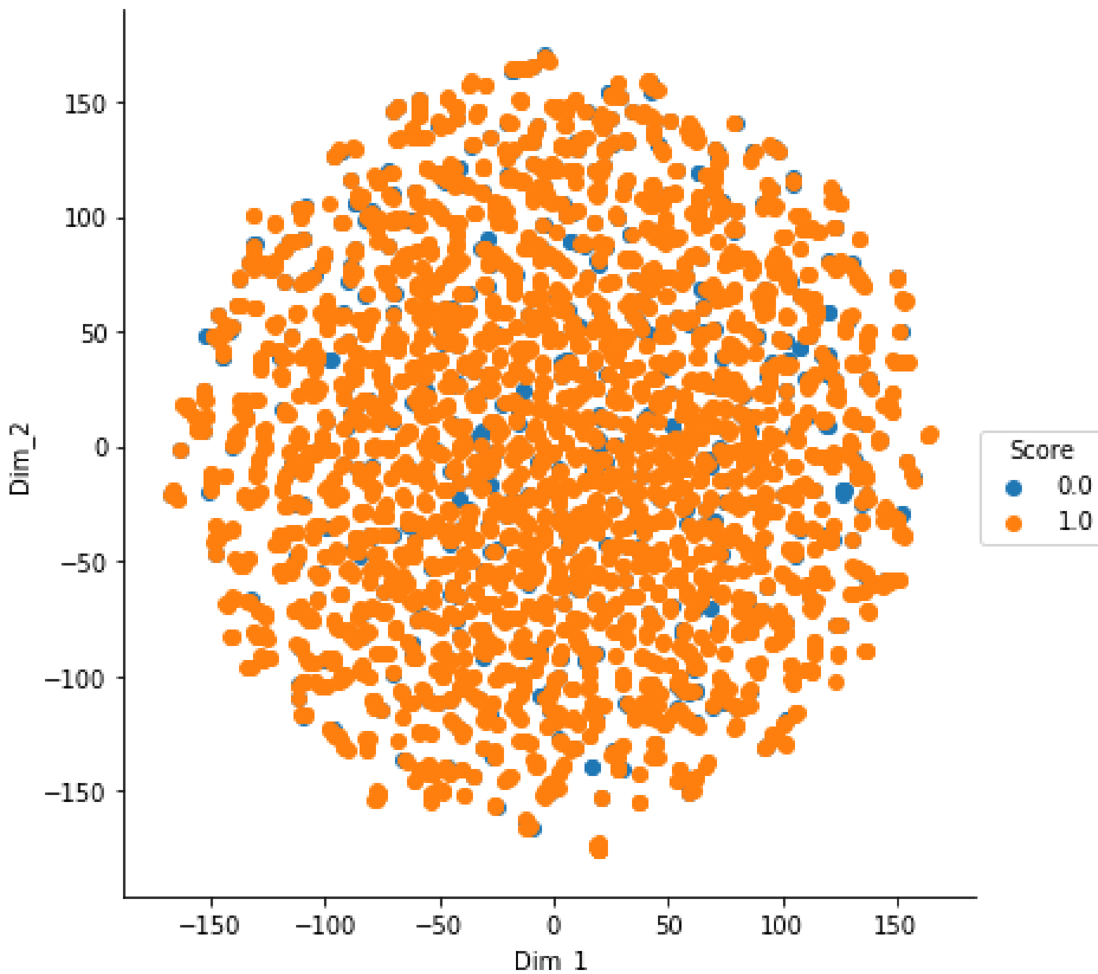
```
tsne = TSNE(n_components=2, perplexity=4000, learning_rate=1500, n_iter=2000)
X_embedding = tsne.fit_transform(standardized_tfidf_w2v_data)
end = time.time()
print('Time Taken:' , end - start)
print(X_embedding.shape)
Y = final['Score']
tsne_data = np.vstack((X_embedding.T, Y)).T
tsne_df = pd.DataFrame(data=tsne_data, columns=("Dim_1", "Dim_2", "Score"))
# Ploting the result of tsne
sn.FacetGrid(tsne_df, hue="Score", size=6).map(plt.scatter, 'Dim_1', 'Dim_2').add_lege
plt.show()
```

```
(4986, 50)
Time Taken: 390.15817499160767
(4986, 2)
```



In [80]:
```
import time
from sklearn.manifold import TSNE

standardized_tfidf_w2v_data = StandardScaler().fit_transform(tfidf_sent_vectors)
print(standardized_tfidf_w2v_data.shape)

start = time.time()
tsne = TSNE(n_components=2, perplexity=2, learning_rate=1500, n_iter=2000)
X_embedding = tsne.fit_transform(standardized_tfidf_w2v_data)
end = time.time()
print('Time Taken:' , end - start)
print(X_embedding.shape)
Y = final['Score']
tsne_data = np.vstack((X_embedding.T, Y)).T
tsne_df = pd.DataFrame(data=tsne_data, columns=("Dim_1", "Dim_2", "Score"))
# Ploting the result of tsne
sn.FacetGrid(tsne_df, hue="Score", size=6).map(plt.scatter, 'Dim_1', 'Dim_2').add_lege
plt.show()
```

```
(4986, 50)
Time Taken: 67.96544599533081
(4986, 2)
```



9 [6] Conclusions

10 Write few sentance about the results that you got and observation that you did from the analysis

Parameters : 1. Tried different Low and High perplexity values from 2 to 2000 2. Tried different values for learning_rate from the default 200 to 4000 3. Number of Iterations was kept mostly at default 1000

Results :

1. The two clusters (Positive and Negative Reviews) can be seen overlapping with each other in most cases
2. The TF-IDF visualization at perplexity=30, learning_rate=500 seemed to provide some separation of the two clusters, but that appears to be because of low learning rate.

Observations : 1. Approaches such as TF-ID weighted W2V can provide the same visualization but with a lot less features than approaches such as BOW and TF-IDF. Because of less number of features the runtime for TSNE is also much better for W2V based inputs.

2. The fact that the number of positive reviews is more can be observed in general (and not necessarily a T-SNE Effect).

3. Most negative reviews appear as close neighbours of some positive reviews, perhaps contaning the same words with just a few words (such as Not) conveying the opposite meaning.

4. With increasing Perplexity the runtime of the T-SNE embedding increases as the algorithm does more work.

5. While OpenTSNE was supposed to work faster, it appears that for small datasets such as 5K its actually slower than the sklearn implementation.